**Author:** Julian Gonzalez

**Assignment:** Lab 3 Report

**Course:** CS 2302 - Data Structures 10:30-11:50

**Instructor:** Fuentes, Olac

**T.A.:** Nath, Anindita

***Introduction***

In this lab we are required to create a balanced BST-tree figure, create a BST-tree with a sorted list, search the BST-tree iteratively, create a list from a BST-tree and finally print all the elements of a BST-tree at a certain depth

***Solutions***

The fist problem had use creating a figure of a binary tree given a list, first was creating a base of when the BST is not none, inside the first thing to be done was to create a circle using the x and y coordinates parameters of the function. The circle was created using the plt.Circle function that was apart of the matplot library of Python. Next thing was to print out the test of the current node, using ax.text function of the matplot library. Next was checking both left and right child of the current node to make sure that they where not none. If this was true it would print the two branches of each child node and do a recursive call with changed parameters eventually printing the entire Tree

For iterative search for BST, I would have a while loop with the condition of the BST not being None inside, I would check if T. item is equal to the given value and if not I would have two other conditions to check if the given value was greater than or less than the current T item, this would insure we checked the correct side of the tree and branches eventually the fist case would be met or we would be left with the ite m not being in the list

For creating a Balanced Binary search tree first, I would create a base case of the length of the given List not equal to one and if not, we would enter the if statement. Inside I created a pivot which would become the root of the tree. Afterwards we could traverse the list left and right sides while adding them into the BST all done recursively.

To create a Sorted list from a balanced tree first I created a base case of when T is not None, inside it I had the function called recursively while traversing the left side only, afterwards the item would be appended into the List we were creating same thing would happen to the right side

To print at a specific depth, base case would be checking if the T is not empty afterwards, I traversed the left and right side of the tree while decreasing the given Depth. Eventually the case if depth would equal 0 printing out the items

***Experimental***

For most of my runs of the programs, I used Tree A = [10,4,15,2,8,1,3,5,9,7,12,18]. For question 3 I created list B = [1,2,3,4,5,6] to be inserted into a new Binary tree, I also tested all the questions with a different list to see if all the functions worked correctly.

A screenshot of a cell phone

Description automatically generatedA close up of an object

Description automatically generated

A picture containing hanger

Description automatically generated***Conclusion***

The lab has taught me how many different things about Binary search trees, the given class code and the functions that I made help me understand them with greater effect

A close up of a map

Description automatically generatedQ1: O(n)

Q2:O(log(n))

Q3:O(n)

Q4:O(n)

Q5:O(n)

Appendix:

1. # -\*- coding: utf-8 -\*-
2. """
3. Created on Thu Mar  7 21:07:47 2019
4. @author: Julian
5. Assignment: Lab 3
6. Instructor: Fuentes, Olac
7. T.A: Nath, Anidita
8. """

11. **import** matplotlib.pyplot as plt
13. **class** BST(object):
14. # Constructor
15. **def** \_\_init\_\_(self, item, left=None, right=None):
16. self.item = item
17. self.left = left
18. self.right = right
20. **def** Insert(T,newItem):
21. **if** T == None:
22. T =  BST(newItem)
23. **elif** T.item > newItem:
24. T.left = Insert(T.left,newItem)
25. **else**:
26. T.right = Insert(T.right,newItem)
27. **return** T
29. **def** InOrder(T):
30. # Prints items in BST in ascending order
31. **if** T **is** **not** None:
32. InOrder(T.left)
33. **print**(T.item,end = ' ')
34. InOrder(T.right)
36. **def** InOrderD(T,space):
37. # Prints items and structure of BST
38. **if** T **is** **not** None:
39. InOrderD(T.right,space+'   ')
40. **print**(space,T.item)
41. InOrderD(T.left,space+'   ')
42. **def** draw\_BinaryTree(ax,x,y,s,w,T):#x,y coordinates, then a scaler s for the whole tree and w for the width of it, T being th BST
43. **if** T **is** **not** None:
44. c= plt.Circle([x,y], 1.5, color='k', fill=False)#uses matplotlib circle function to draw a circle for each node
45. ax.add\_artist(c)#adds the circle into the figure
46. ax.text(x-.7, y-.5, T.item, size=6) #prints the valute of the current node
47. **if** T.left **is** **not** None:#prints the left side of the tree
48. xL=[x,x-(s\*s)]
49. yL=[y-1.5,y-w]
50. ax.plot(xL,yL,color='k')
52. draw\_BinaryTree(ax,x-(s\*s),y-w-1.5,s-1,w,T.left)
53. **if** T.right **is** **not** None:#prints the right side of the tree
54. xR = [x,x+(s\*s)]
55. yR = [y-1.5,y-w]
56. ax.plot(xR,yR, color='k')
57. draw\_BinaryTree(ax,x+(s\*s),y-w-1.5,s-1,w,T.right)
59. **def** IterSearch(T,k):
60. **while** T **is** **not** None:
61. **if** k == T.item:## if the current node of the tree is k return current node T
62. **return** T
63. **elif** k > T.item:#checks if k is greater than current node item if it is continues to the right of the tree
64. T = T.right
65. **else**:#if all other checks fail it will defualt to looking in the left tree
66. T = T.left
67. **print**('Could not find k inside the list, inserting k into list')
68. T = Insert(T,k)# doesnt actually insert into the main list
69. **return** T
71. **def** insertBBT(B):
72. **if** len(B) != 0:#checks the length of the B list if its 0 it returns
73. pivot = len(B)//2#The mid point of the list evertime used for inserting at every recusive call
74. temp = BST(B[pivot])#creates the binary tree with the root being pivot
75. temp.right = insertBBT(B[pivot+1:])#call to insert to the right of the binary tree
76. temp.left = insertBBT(B[:pivot])#call to insert to the left of the Binary tree
78. **return** temp #returns the completed tree
80. **def** extractor(T,List):
81. **if** T **is** **not** None:
82. extractor(T.left,List)### traverses all the way to the left side of the tree(smalles varible)
83. List.append(T.item) #adds the current node item into the list
84. extractor(T.right,List)### starts to traverse the right side of the tree then adds on the the list
86. **def** printAtDepth(T,d):
87. **if** T **is** **not** None:
88. **if** d == 0:# prints the items at depth d
89. **print**(T.item)
90. **else**:
91. printAtDepth(T.left,d-1)#goes to the left subtree to print items at depth d
92. printAtDepth(T.right,d-1)#goes to the right subtree to print items at depth d
93. # Code to test the functions above

96. T = None
97. A = [10,4,15,2,8,1,3,5,9,7,12,18]
98. **for** a **in** A:
99. T = Insert(T,a)
100. #####################################
101. **print**('------------------------------------------')
102. **print**("Question 1")
103. #1)
104. **print**("Printing Binary Tree...")
105. fig, ax = plt.subplots()
106. draw\_BinaryTree(ax,0,0,4.5,15,T)
107. ax.set\_aspect(1.0)
108. ax.axis('off')
109. plt.show()
111. ######################################
112. **print**('------------------------------------------')
113. **print**("Question 2")
114. #2) Iterative search
115. s= 15 ## change to find desired value
116. x = IterSearch(T,s).item
117. msg = 'Looking for ' + repr(s) + ': ' + repr(x)
118. **print**(msg)
120. #####################################
121. **print**('------------------------------------------')
122. **print**("Question 3")
123. #3) Build a balanced  binary tree with sorted list
124. B = [1,2,3,4,5,6]
125. T2 = insertBBT(B)
126. **print**("Printing Binary Tree...")
127. fig, ax = plt.subplots()
128. draw\_BinaryTree(ax,0,0,4.5,15,T2) #DONT FORGET TO USE DRAG TO TO LOOK AT WHOLE FIGURE
129. ax.set\_aspect(1.0)
130. ax.axis('off')
131. plt.show()
133. #####################################
134. **print**('------------------------------------------')
135. **print**("Question 4")
136. #4)Extract elements of BST into a sorted list
137. emptyList = []
138. extractor(T,emptyList)
139. **print**(emptyList)
141. ####################################
142. **print**('------------------------------------------')
143. **print**("Question 5")
144. #5)Print depths
145. depth = 2 ##choose which depth to print out
146. msg2 = 'Keys at depth ' + repr(depth) + ': '
147. **print**(msg2)
148. printAtDepth(T,depth)

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

* Julian Gonzalez